An optimized stereo vision implementation for embedded systems: application to RGB and Infra-Red images

Real-Time Image Processing manuscript No.
(will be inserted by the editor)

Simone Madeo · Riccardo Pelliccia · Claudio Salvadori · Jesus Martinez del Rincon · Jean-Christophe Nebel

An optimized stereo vision implementation for embedded systems: application to RGB and Infra-Red images

Received: date / Revised: date

Abstract The aim of this paper is to demonstrate the applicability and the effectiveness of a computationally demanding stereo matching algorithm in different low-cost and low-complexity embedded devices, by focusing on the analysis of timing and image quality performances. Various optimizations have been implemented to allow its deployment on specific hardware architectures while decreasing memory and processing time requirements: (1) reduction of color channel information and resolution for input images, (2) low-level software optimizations such as parallel computation, replacement of function calls or loop unrolling, (3) reduction of redundant data structures and internal data representation. The feasibility of a stereovision system on a low cost platform is evaluated by using standard datasets and images taken from Infra-Red (IR) cameras. Analysis of the resulting disparity map accuracy with respect to a full-size dataset is performed as well as the testing of sub-optimal solutions.

Keywords Stereo-vision · Embedded optimization · Embedded Systems · Smart Camera · Near Infra-Red

Simone Madeo
TeCIP Institute, Scuola Superiore Sant'Anna,
56124 Pisa Italy
Tel.: +39 050 5492037
E-mail: simone.madeo@sssup.it

Riccardo Pelliccia
TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: riccardo.pelliccia@sssup.it

Claudio Salvadori
TeCIP Institute, Scuola Superiore Sant'Anna, Pisa, Italy
E-mail: claudio.salvadori@sssup.it

Jesus Martinez del Rincon
The Institute of Electronics, Communications and Information Technology (ECIT), Queen's University of Belfast
BT3 9DT, United Kingdom
E-mail: j.martinez-del-rincon@qub.ac.uk

Jean-Christophe Nebel
Digital Imaging Research Centre, Kingston University
KT1 2EE, United Kingdom
E-mail: j.nebel@kingston.ac.uk

1 Introduction

Stereo vision systems deal with the extraction of three-dimensional information in scenes captured from two different viewpoints by comparing the relative positions of objects. Automatic procedures enabling stereoscopic perception have been used in a wide range of applications, such as visual robot navigation, cartography, aerial reconnaissance, video surveillance and close-range photogrammetry. These techniques are also of great interest in tasks such as image segmentation for object recognition and the construction of multi-dimensional scene models in image-based rendering [20].

Stereo matching algorithms are widely recognized as computationally expensive processes [30], since they involve demanding problems such as conjugate pairs detection in the input images in order to generate the disparity map. By assuming a system where the two cameras share the same image planes, the horizontal displacement is used to obtain the depth of objects in the scene. In recent years several hardware solutions have been explored by the community, considering the use of dedicated architectures that enable parallel operations [1] and [35], and the exploitment of real-time capabilities for high-performance embedded systems [31] and [66]. Among the possible options, microcontrollers and microprocessor-equipped embedded boards provide some advantages, such as easiness in designing, simulating and integrating different peripherals, flexibility in reprogramming using Flash, EEPROM or EPROM, and affordability for implementation [58]. Although these benefits allow extraordinary pervasiveness and availability of such devices, these boards are characterized by severe memory and processing constraints.

In order to address these challenges, compromises must be taken to embed a stereo matching algorithm in a specific device. Analysis of how this affects both performance and applicability of the algorithm is the focus of the proposed work. Exploiting a stereo matching algorithm [14], able to provide an excellent com-

promise between speed and accuracy, various optimizations have been implemented to allow its deployment on specific hardware architectures while decreasing memory and processing time requirements: (1) reduction of color channel information and resolution for input images, (2) low-level software optimizations, (3) analysis of the resulting disparity map accuracy with respect to a full-size RGB dataset and (4) testing suboptimal solutions. In addition to implementation issues, image analysis of real scene suffers from variations in terms of brightness and contrast, in particular in night environments. In this work, the use of only luminance component of the input images in the stereo matching procedure has been investigated to measure whether a significant deterioration in the resulting disparity map accuracy has been introduced. The correctness of this hypothesis allows to demonstrate the feasibility of a stereovision system able to process images even taken from Infra-Red (IR) cameras.

The remainder of the article is organized as follows. In Section 2, a summary of the state-of-the-art solutions addressing the stereovision problem, in the presence of memory and processing power constraints, is introduced. Section 3 describes the contributions of this paper. In Section 4 the stereo matching algorithm used in this work is presented, with a particular focus on its requirements in terms of memory and computational resources. Proposed software optimizations are discussed in Section 5. In Section 6 a performance analysis for a range of devices and experimental results is presented using standard datasets and an application to night vision. Conclusions and future works follow in Section 7.

2 State of the art

In order to face the problem of finding correspondences between pixels belonging to a pair of stereo images, a wide range of techniques have been proposed, such as block correlations, dynamic programming, graph cut and simulated annealing. Comprehensive reviews of the literature can be found in [44] and [56]. Besides quality performance of such a wide variety of techniques, time and memory requirements of a stereo matching algorithm are critical aspects to be evaluated prior to implementation. There are two main ways to deal with these requirements: exploiting *software optimizations* by using specific programming methodologies, and *hardware capabilities* of dedicate units such as FPGAs or GPUs.

Regarding software optimizations, in [61] the authors propose block-based belief (BP) propagation to reduce the required memory size by 99% compared to the original implementation of the stereo correspondence algorithm. It relies on a graph model that contains nodes representing pixels and connected edges representing pipes in a Markov network. The algorithm is based on a flow diagram of messages passing along the 4-connected edge iteratively until a cost function converges. The graph model is partitioned into $B \times B$ block: this enables parallelism, earlier convergence for each block, and lower memory size requirement. A typical belief propagation algorithm needs 5 WHL memory units, where W and H are the dimensions of the image and L the range of disparity. The block-based version requires only $5B^2L$, but leads to a slightly increased error rate by $2\% \sim 3\%$ compared to a typical belief propagation. The main issue of this approach is the average number of iteration for convergence in different block size. For block size 16×16, the minimal average iteration count is about 30, but this will suffer large error rate with significant blocky effect. Bigger block size enables better quality but slower performance.

Markov Random Fields (MRFs) are well known models used in many low-level Computer Vision applications dealing with energy minimization in multi-label classification problems, i.e. given a set of labels Γ, learning from a set of examples that are associated with a subset of Γ. An exploitation of the regularity of simple energy functions is required to avoid exhaustive search of label space. The LogCut algorithm [37] explores the label space, represented by disparities, by binary subdivision applied at successive bit-levels, starting from the most significant. Time complexity of this method is logarithmic in the size of the label set rather than linear. The selection of the most significant bits requires the reiteration of the algorithm by using different hierarchical partitions of the label space, corresponding to different bit codings of the labels. These different solutions are combined in an optimal way via additional graph cut operations. The K-dimensions label space is partitioned according to values of binary bits $b = 1, \ldots, B$, with $B = \log K$, starting with the most significant, and the labeling of a given pixel can be achieved by executing a tree of binary classification steps. Respect to linear approaches as α-expansion [6], a speed-up of order $K/\log K$ is obtained. In return for the advantage in computational efficiency, LogCut requires an additional offline learning step to compute the values of unary potentials for MRF. Therefore, main issues are related to provision of suitable training data, and practical limitations on working memory for large training sets and large label spaces.

Regarding software implementation of stereo matching techniques, those based on dynamic programming (DP) provide good accuracy and are computationally efficient [40], since the global minimum for independent scanlines can be found in polynomial time. This feature makes them specially attractive when considering their implementation in resource-constrained devices by using parallel programming. Sparse disparity maps can be achieved by using edge information [2] and [47], while dense maps need correspondences between scanlines by using pixel color values. Some issues deriving from this approach include indistinct image features, image noise and half occlusion. Solutions based on statistical estima-

tion, also able to address vertical consistency between scanlines, are presented in [24] and [60]. Another class of DP algorithms [12] and [62] explores vertical consistency by using a tree structure, as opposed to the individual scanlines. Results show an improved accuracy with only a marginal increase of computational cost. In the last decade, the main emphasis has been on designing real-time solutions by adapting previous DP algorithms [19], [53] and [64]. Performance achieved by [40] demonstrates DP-based approaches provide the best compromise between accuracy and speed.

The aim of achieving optimal results in terms of speed and accuracy depends on the nature of the algorithm as well as the hardware platform. Real-time implementations characterized by low errors compared to the ground truth require heavy computational power. As a result, decent sized set of images cannot be used to achieve the required disparity precision using standard sequential processing methods. One promising direction towards achieving real-time performance in high performance computing applications such as image processing would be to exploit the parallelism in general purpose GPUs [26] and [65]. In [8] the stereo vision system is obtained by integrating Gabor filter based on biologically motivated algorithms to CUDA (Compute Unified Device Architecture) using an API by Nvidia which could be used to directly access the GPU. The Single Instruction Multiple Data (SIMD) architecture allows the same activity being carried out simultaneously in thousands of threads over different blocks of data by involving the usage of an efficient memory hierarchy module. GPU's *pinned* memory cannot be swapped to disk and provides improved transfer speeds respect to memory allocated using the `malloc` function. The transfer is asynchronous meaning that the host would not wait upon the completion of the transfer to move forward. The process is optimized by pre-loading a window of the image to the Shared memory, and all the stored pixel values are reused by all the threads in the same block. The process has been sped up to 77 fps.

FPGA-based stereo vision systems have also been introduced due to the rapid development of programmable devices [35]. Even though considerable progress has been made, improvements are still needed in terms of architecture. These aspects are related to the stereo matching methods, frame rate requirements, scalability, and one-chip integration of pre- and post-processing functions. An example is given by the work in [13] dealing with the design and the implementation of a general architecture characterized by five main modules: the acquisition of original left and right images, the rectification, the mean operator, the Census transformation and the Census correlation. Images are not stored completely and pixels are processed on the fly, with a pixel clock defined by the cameras. The rectified left and right pixels are generated with a delay of N_{buf} lines with respect to the acquisition. Two mean operator modules apply a mean filter independently on the two images, based on a window centered around every pixel. The Census Transformation (CT) modules compute for each pixel a bitstring from comparisons with their neighbors. The correlation scores are computed and maximum scores are selected by a Census Correlation module. Given the maximal disparity in pixel D_{max}, the scores computation is parallelized to match left pixels with right ones by using a SIMD operation, so that $D_{max} + 1$ identical and synchronous processes can provide all scores in parallel. When the left CT pixel (u, v) is available, its corresponding one in the right image is already computed, while a verification is made using right-left scores and right-left matching. The final latency between the original acquisition and the generation of the disparity for a (u, v) pixel, can be approximated by $N_{buf} + k$ lines, where k is a constant depending on both the distortion factor of the original images and the window size used to compute the mean operator and the Census Transformation. The disparity map is sent at the same frequency than the original images and the process has been sped up to 160 fps.

Although both GPUs and FPGAs are able to provide very high-rate stereo correspondences, the drawbacks are obviously related to the cost of those boards. For instance, boards used in [21] are Stratix III E260 with ≈100000 logic elements (LEs) and NVIDIA GeForce 295 GTX. Their cost is 2450$ and 370$ respectively. Although much cheaper FPGA development kits (e.g. DE0 nano with 22320 LEs: 79$) can be found on the market, graphic processing needs a higher number of LEs (e.g. ≈70000 LEs in [50]). GPUs are characterized by a lower cost, but a higher power consumption (275W vs 100W for FPGA [21]) which could not be feasible for embedded computing. Regarding simple microcontroller boards, a reference price can be 200$ [57] per unit and only 14$ [42] for the single microcontroller: as for other custom electronic boards, a high scalability of the price in case of large quantities is achieved. A common low-cost single-board computer development platform costs 202$ [49].

As for what concerns development complexity, FPGA boards are commonly based on synthesizable VHDL models. A larger number of concurrent VHDL statements and small processes connected through signals are used to implement the desired functionality. Reading, understanding (and obviously modifying) dataflow VHDL code is difficult since the concurrent statements and processes do not execute in the order they are written, but when any of their input signals change value [23]. GPUs have issues related to compatibility and portability, since vendors support different APIs, e.g. CUDA or OpenCL, and performance improvement could be highly variable depending on the type of graphic card (integrated, proprietary). Moreover, not all tasks are suited for running on a GPU: I/O-bound operations and direct access to system memory are the main examples.

3 Contributions of this paper

The aim of this paper is to demonstrate the applicability of a computationally demanding stereo matching algorithm in different low-cost and low-complexity embedded devices, by focusing on the analysis of timing and image quality performances. The feasibility of data reduction methodologies or programming methodologies such as parallel processing, memory and computational optimizations, is the keypoint enabling an efficient implementation of the proposed stereo matching algorithm even in low-power vision-enabled sensor networks. Distributed stereo processing and coding [9] for surveillance are promising fields of study: smart camera motes forming a vision-enabled network can add increasing levels of intelligence [28]. Moreover, by using precompiler directives, the source code has been kept as generic as possible in order to make the porting of the algorithm simpler.

The implementation framework is addressed as follows. First, reduction of the amount of data to be processed in terms of both image resolution and representation (i.e. the number of channels per pixel) has been investigated. Second, the code has been optimized in order to reduce its computational cost by using standard code optimization techniques (parallel computation, replacement of function calls, loop unrolling), and its memory footprint by reducing redundant data structures and internal data representation. Third, stereo matching correspondences have been computed by means of a simplified procedure which does not involve vertical discontinuities analysis.

Regarding memory optimizations, by exploiting the way the internal data structure is used to find correspondences, up to an order of magnitude of memory can be saved. Regarding data reduction analysis, the resolution of the input images can be decreased by applying a factor γ_w and γ_h to width and height respectively. For instance, halving both width and height of the original images, that is $\gamma_w = 1/2$ and $\gamma_h = 1/2$, results in a new input images reduced by a factor $\gamma = \gamma_w \cdot \gamma_h = 1/4$. The number of channels per pixel can also be reduced by passing from an RGB color mode to a grayscale one. However, such data reduction has an implicit cost in terms of accuracy that needs to be considered and analyzed.

Moreover, image processing using only the luminance component, i.e. the Y channel in the YUV color space, allows the use of acquisition systems equipped with *Near Infra-Red* (or *NIR*) sensors and the implementation of stereo vision systems able to work in a night environment. Results obtained by NIR images have been evaluated in order to compare with performances achieved in the RGB domain. The effects of different light condition, during the day with natural light or during the night with NIR cameras, have also been analyzed in order to demonstrate the independence of gray levels distribution in the grayscale stereoscopic image with respect to the brightness of input image pairs, and the applicability of the proposed implementation, even the most constrained versions, in real environment.

4 The algorithm

The stereo matching algorithm proposed by [14] and originally derived from [45] is particularly suitable for the scanline to scanline correspondence problem, which can be applied to pairs of rectified stereo images. Another promising version, even able to face distortions, is described in [15] while a suitable implementation of the algorithm on FPGA can be also found in [63].

Following a dynamic programming approach [27], sequences of data can be compared to provide an effective automatic method to produce an exact solution to the global alignment of character strings according to a scoring function, taking into account possible mutations in the sequences. Alignments are produced by first filling in a scoring matrix containing the local alignments, and then *backtracking* from the highest score in either the last column or the last row of the matrix in order to extract the global alignment.

The proposed work can be considered the first stage for the realization of an extended DP version [41] able to deal with unrectified and non-linearly distorted images. By using a larger dimensional space and a 3D scoring matrix, correspondences between a line and a whole image can be also calculated.

4.1 The scoring matrix

The scoring matrix E is filled in using scoring functions which quantify the similarity of possible pairings. The matrix is initialized by setting the value in the top left cell to zero and the first line and column according to cumulative gap penalties. Each matrix cell stores the maximum value which can be achieved by extending the previous alignment up to that point. This can be done either by aligning the next pixel of the first sequence with the next pixel of the second sequence or extending a sequence by a special value to record a pixel insertion or deletion. In the case of pixel alignment, i.e. diagonal motion in the matrix, the score depends on their values. A reward, *match*, is allocated if the two pixels are identical, otherwise a penalty, is applied since this highlights a mutation or substitution. The mismatch penalty of aligning a pair of pixels, where p_i and p_j are their values, is expressed by the absolute value of their difference, so that extending an alignment along the diagonal alters the global score by the following:

$$\Delta_{i,j}^{diag} = match - |p_i - p_j|. \quad (1)$$

When a sequence is extended, i.e. from either north or west, this is also penalized by *gap*, since it reveals that a mutation as insertion or deletion occurred. Due to

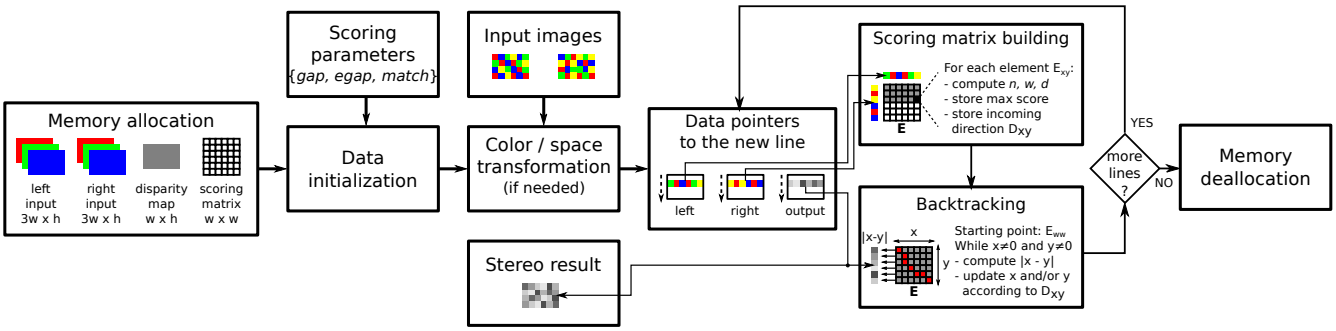


Fig. 1: Data-flow model of the algorithm. After memory allocation and data initialization, for each input image pair of lines *scoring matrix building* and *backtracking* modules are applied.

the nature of stereo matching, different camera viewpoints create occlusion areas associated with each object present in a scene. Assuming that a few occlusions of several-pixel length would be more frequent than a large number of 1-pixel occlusions, a lower penalty for extended gaps, *egap*, has also been introduced to encourage gaps to cluster. The whole procedure to fill in the scoring matrix E is shown in Algorithm 1.

Algorithm 1 Filling of the scoring matrix.

Input: *gap*, *egap* and *match* integer values; l and r two sequences of characters.
Output: scoring matrix E.

$north = 0$; $west = 0$; $diagonal = 0$;
for each row i in the first sequence **do**
  for each column j in the second sequence **do**
    $mismatch = -\text{ABS}(l[i] - r[j])$;
    if $E[i-1, j]$ is a gap **then**
      $north = E[i-1, j] + egap$;
    else
      $north = E[i-1, j] + gap$;
    end if
    $diagonal = E[i-1, j-1] + match + mismatch$;
    if $E[i, j-1]$ is a gap **then**
      $west = E[i, j-1] + egap$;
    else
      $west = E[i, j-1] + gap$;
    end if
    $E[i, j] = \max(north, diagonal, west)$;
  end for
end for

4.2 Backtracking

While completing the matrix, in addition to the score of each cell, the direction from which the score is coming must be recorded in order to perform the backtracking process and extract the optimal alignment. This procedure usually produces a set of optimal alignments. Consequently, new information needs to be supplied to allow selecting a single solution [29], [36] and [46].

	-	5	7	1	3	5
-	0	→ -1	→ -2	→ -3	→ -4	→ -5
5	↓ -1	↘ 2	→ 1	→ 0	→ -1	↘ -2
6	↓ -2	↓ 1	↘ 3	→ 2	→ 1	↘ 0
3	↓ -3	↓ 0	↓ 2	↘ 3	↘ 4	→ 3
4	↓ -4	↓ -1	↓ 1	↓ 2	↘ 4	↘ 5

Fig. 2: Completed scoring matrix and optimal path highlighted. Candidate direction values are in red.

Several strategies have been offered to deal with this issue in the context of stereo matching. Many suggest selecting the 'smoothest' solution in term of horizontal and vertical discontinuities along and across scanlines [5] and [10]. Some are based on high confidence matches, such as edge intersections, which are identified during a pre-processing phase. These good matches are exploited as extra constraints in the choice of a unique solution [5] and [60]. In [14] the traditional bioinformatics approach has been followed: each scanline can be seen as a mutation of both the previous and the following lines. Therefore, alignments involving these lines can be used to select among several solutions by enforcing some vertical discontinuities.

The whole process for two sequences `57135` and `5634` is illustrated in Figure 2. The following scoring scheme[1] has been used: $match = 2$, $gap = -1$ and $egap = -1$. To perform the backtrack procedure, the highest score cell in either the last column or row is identified, in this case the bottom right corner of the matrix, i.e. $e_{4,5}$. The following candidate values can be computed: from north $e_{3,5} + gap = 2$, from west $e_{4,3} + gap = 3$ and from north-west $e_{3,3} + \Delta_{4,5}^{diag} = 5$. The highest value, in this case from north-west, is stored: $e_{4,5} = 5$. Then, using direction information, a path to the origin of the matrix is constructed. Finally, this path is converted into the fol-

[1] The digits in the sequences can be seen as pixel values of a given channel in the input image pairs or, more in general, as elements to align. For the sake of simplicity, by stating $gap = egap$, effects of extended gap have not been taken into account.

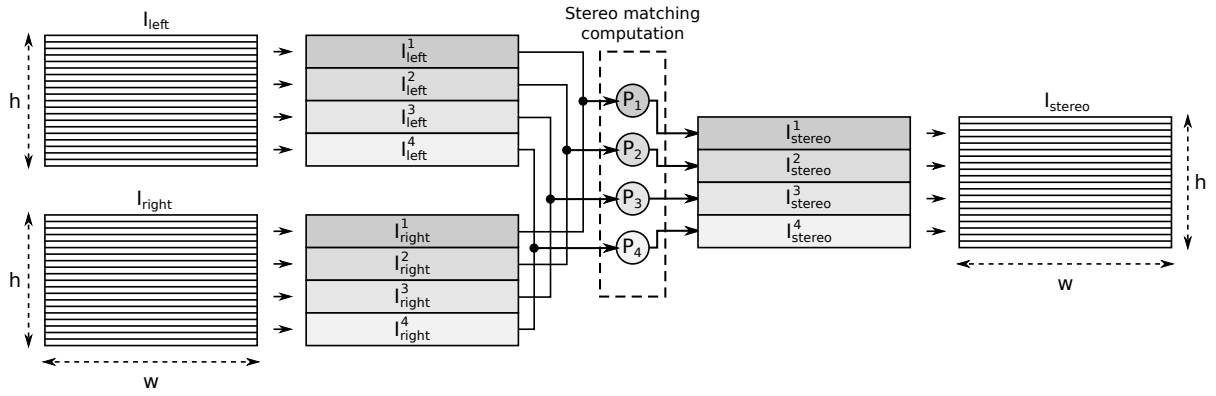


Fig. 3: A 4-core stereo pipeline for bursts of lines. The input image pairs I_{left} and I_{right} are divided in 4 bursts of lines, according to the number of cores. Each core P_i processes in parallel their own bursts I^i_{left} and I^i_{right}, and computes line by line the resulting burst of disparities I^i_{stereo}.

lowing resulting alignment:

```
57135
56-34
```

In the alignment gaps are represented by '-'.

5 Implementation

As explained in Section 4, previous algorithm [14] is not only providing good results in stereo matching, but it also comprises some ideal properties to be implemented by parallel computation. By reducing the global alignment to scanline alignments, only one line has to be considered at a given time and both memory requirements and computational complexity can be reduced by number of lines.

In order to embed the algorithm in boards with different characteristics, such as maximum CPU clock rate and available RAM, some coding optimization techniques have been taken into account. These techniques can be divided in two categories [48]: *processing optimizations* aims to minimize the overall processing time; *memory optimizations* aims to reduce the total amount of memory required to execute the algorithms.

In the remainder of this article the Big O notation $O(\cdot)$ will be used to represent the complexity of an algorithm as a function describing its efficiency in terms of the amount of data to process. Both *time complexity* and *space complexity* can be defined to estimate the amount of time and memory an algorithm takes according to the input size [38].

5.1 Processing optimizations

Major processing optimization deals with parallel computation, function stack management and loop cycles simplification [25], as described below. Others involve reformulating mathematical formulas, such as replacing multiplications with addition, whenever possible, and privileging temporary variables usage in order to reduce memory accesses.

5.1.1 Parallel computation

Since stereo lines are computed independently from each other, each left and right stereo images of resolution $w \times h$, where w is the width and h the height in pixel, can be split in bursts of lines and executed in parallel on N different processors $\{P_1, P_2, \ldots, P_N\}$ as shown in Figure 3. By defining a parameter $\alpha = h \bmod N$, the first α cores process a number of lines equal to L_α, while the remaining $N-\alpha$ cores process a number of lines equal to $L_{N-\alpha}$, e.g. $L_\alpha = \lceil \frac{h}{N} \rceil$ and $L_{N-\alpha} = \lfloor \frac{h}{N} \rfloor$ respectively. In case $\alpha \neq 0$, the first α processors compute one stereo line more than the others. This strategy for the allocation of bursts of lines to the available processors is the simplest to implement and the fastest to perform. The processing time to obtain the resulting stereo line for each couple of input lines may vary, depending on the distribution of colors in the input images, the building of score matrix and the subsequent backtracking procedure. A dynamic allocation module should be able to assign, at the sampling time $\bar{t}$, each line $l_i^{left}(\bar{t})$ and $l_i^{right}(\bar{t})$ for $1 \leq i \leq h$, to a processor P_j for $1 \leq j \leq N$, and to keep track of the CPUs load. Without any *a priori* knowledge about the input data, the implementation of a dynamic allocation is not recommended. A processing overhead is required to scan the current line pairs, i.e. $O(w)$ time complexity, and perform a heuristic able to estimate the time required to get the resulting stereo line. Benefits from parallel programming will be described in Section 6.5, while the increasing memory demand is explained in Section 5.2 and shown in Eq. (6).

5.1.2 Replacement of function calls

A call stack is mainly used to keep track of the point at which an active subroutine should return control at the end of its execution. Function calls cause a significant overhead in the case of intensive computation tasks [59], since the call stack is in charge of managing the local variables for the invoked subroutine, the return address and all the parameters required by the function. Two important subroutine executions in the proposed algorithm are the call of 1) the *mismatch value* computing method and 2) the *backtrack* module. The first function occurs w^2 times for each scanned line, since the mismatch value is computed for each element of the score matrix, the second one occurs at least w times and not more than $2w$ times, depending on the backtracking direction priority, as explained in Section 5.2.5.

To prevent this large amount of subroutine calls, those routines were implemented within the body of #DEFINE C preprocessor instructions rather than as functions. The instructions avoid the use of the stack by copying routine codes where needed. Code duplication in the *text segment* of the memory is not an issue since the body of the routine is small: GNU GCC compiler [32] produces 25 assembler instructions for the mismatch value function and 45 for the backtracking procedure. By enabling #DEFINE instruction the code size increases by 25.4%.

It is recalled that preprocessor macros are preferable to *inline* functions: depending on the specific configurations for the used compiler and the release builds, these functions are not always guaranteed to be inlined. For instance, MPLAB® XC compilers for PIC32 offer different level of optimization depending on the Edition Types [43].

5.1.3 Loop unrolling

Loop unrolling is a well-known program transformation able to reduce the increment-and-test overhead for loop iterations [55]. The program speed can be increased by reducing the number of instructions that control the loop, reducing branch penalties and the memory reading delay. To eliminate this overhead, loops can be re-written as a repeated sequence of similar independent statements. For modern processors, the primary benefits of loop unrolling include increased instruction-level parallelism, improved register locality and memory hierarchy locality. This technique also allows amortizing the overhead of a single prefetch instruction across multiple load or store instructions. In the code below a simple 8-instruction FOR loop unrolling in C language is shown.

```
#define UNROLL_8(expr)
    expr; expr; expr; expr;
    expr; expr; expr; expr;
```

The body of the loop, *expr*, is repeated 8 times and contains all the instructions to reiterate. The code above is the first step to build more generic routines, such as a K-instruction FOR loop. For example, if $K = 34$, then a `UNROLL_8()` routine can be executed four times, plus two residual cycles. The declaration of a generic unrolled loop is the following:

```
#define UNROLL_8_GENERIC(cond, r, expr)
```

In this case the routine *expr* also contain the code able to change the exit condition, i.e. *cond*. If $K = 34$, then a program starting with a loop counter $i = 0$ will have an instruction `i++` in the body of the loop and the statement `i < 32` as exit condition. The residual amount r in a `UNROLL_8()` loop cycle is computed by the programmer:

$$r = K - \left(8 \cdot \left\lfloor \frac{K}{8} \right\rfloor\right). \tag{2}$$

The proposed `UNROLL_32()` loop unrolling routine has been compared with standard GCC loop optimization. Experiments show almost identical timing results at 10 ns precision for both techniques by using the dataset cited in Section 6.3 as input. Compared to the use of normal loops, the time saved is 4%. Since GCC compiler is not available for all the boards tested in this work, as explained in Section 6.2, the proposed loop unrolling technique has been used as generic code feature for further experiments.

In case of longer FOR loop cycles, other routines such as `UNROLL_32()` or `UNROLL_64()` can be implemented. Given n_{call} the number of times a K-instruction loop is present in the source code, by using loop unrolling the body of the loop in the code segment memory will increase by a factor $n_{call} \cdot K$. In the proposed implementation, the use of `UNROLL_32()` routine causes an increase in code size by 400%.

5.2 Memory optimizations

In order to evaluate the amount of required memory, let assume the followings: the parameter c is the number of input image channels (3 for RGB images, 1 for 8-bits grayscale images[2]), while ε is the size in bytes of a single score matrix element $e_{i,j}$, for each $i, j \in [1 \dots w]$. Without considering the negligible size of a few local variables, the memory in bytes used during the execution of the proposed algorithm is composed by the followings:

- left input image ($w \times h \times c$);
- right input image ($w \times h \times c$);
- stereo output image ($w \times h$);
- score matrix ($N \times w \times w \times \varepsilon$).

[2] For Near-Infrared images a grayscale color encoding is assumed.

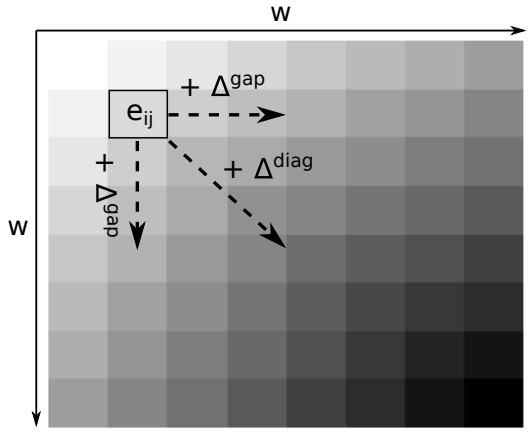


Fig. 4: Score matrix building.

At the first stage of memory optimization, the score matrix E is a reusable memory area managed by a single processor, that sequentially computes new score values for each pair of input lines and overwrites the old ones when no longer needed. When using N processors, a dedicated memory area for each core is required.

5.2.1 Score value dimensioning

According to the default values in the previous work [14], without any *a priori* knowledge about the analyzed images, let assume for gap and extended gap parameter the following values:

$$gap \leq egap \triangleq \Delta^{gap} \leq 255. \tag{3}$$

The dimensioning procedure for these parameters is an important step to correctly size the required memory for the implementation of the proposed algorithm. Without loss of generality *gap* and *egap* values can be considered as natural numbers coded in $\log_2(255+1) = 8$ bits. During the score matrix building procedure, $e_{i,j}$ increases in both x and y directions by either *gap* or *egap* and in diagonal direction by the following maximum value:

$$\Delta^{diag} = \max_{i,j}\left(\Delta^{diag}_{ij}\right) \leq 255. \tag{4}$$

If $\Delta^{gap} < \Delta^{diag}$, in case of perfect matching in the input images, the chosen score value is always in diagonal direction, as shown in Figure 4. Starting from the top-left corner, the path will end in the bottom-right corner, following the diagonal of the matrix, by adding Δ^{diag} for each line. The maximum score value can fit in n_{max} bits:

$$n_{max} = \log_2\Big(w \cdot \Delta^{diag}\Big) = \log_2\Big(w \cdot 255\Big). \tag{5}$$

Using Equation (5), if $w = 450$ then $n_{max} \approx 17$ bits; for images with bigger resolution, as $w \approx 1000$, n_{max} is always fewer than 21 bits.

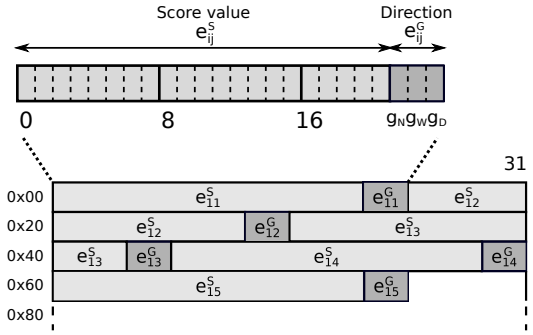


Fig. 5: A 24-bits score matrix element $e_{i,j}$ and a 32-bit memory alignment.

5.2.2 Score matrix reduction

In order to optimize the required memory further, bitwise operations have been adopted for low-memory systems. Three bits, g_n, g_w and g_d, are used to represent the gap direction used for the backtracking procedure: if $g_n = 1$ there is a gap in the *north* direction, if $g_w = 1$ there is a gap in the *west* direction and if $g_d = 1$ there is a gap in the *diag* direction. If the size of $e_{i,j}$ is 24 bits, the three parameters g_n, g_w and g_d can be easily stored in the last three bits of each $e_{i,j}$ element, by using the bit-shift operators, as '`<<`' and '`>>`', and by applying the appropriate bit masks. Consequently, in the proposed implementation the size of $e_{i,j}$ is 24 bits, i.e. $\varepsilon = 3$, containing both information about mismatch values and gap directions, as shown in Figure 5. Every $e_{i,j}$ element is divided in a score value part $e^S_{i,j}$ and in a direction part $e^G_{i,j}$. Considering a 32-bit architecture, the memory alignment depicted in Figure 5 should be used: each $e_{i,j}$ element is located in a contiguous area of the memory reserved for the E matrix. In this way not even a single bit will be wasted. Since $e_{i,j}$ memory area can be in between two 32-bit memory lines, this operation has an impact on the number of memory accesses in order to get or set a score value. Experiments show that `get()` and `set()` operation time has increased by 75% and 25% respectively.

5.2.3 Resolution reduction

The total amount M of memory in bytes[3] required by the program is:

$$M = w \cdot \Big(2h \cdot c + h + N \cdot w \cdot \varepsilon\Big). \tag{6}$$

For example, since the *Cones* pair of color images from Middlebury dataset [56] has a 450×375 resolution, the

[3] The total amount M has not been rounded to the upper multiple of 32 since other small data structures not considered in Equation (6) can be used to keep the alignment and save memory. Moreover, the unrounded version M is useful to have a more accurate comparison of memory requirements in case of use of different datasets.

size of the uncompressed RGB image is 506250 bytes. In case of a single-core evaluation, $M = 1788750$ bytes. By considering the same images with Quarter-QVGA (QQVGA) 160×120 resolution, usually used in displays of handheld devices, the algorithm requires $M' = 134400$ bytes in case of single channel. Assuming a $w' \times h'$ notation for low resolution images, the compression factor η for the required memory is the following:

$$\eta = \frac{M}{M'} = \frac{w \cdot \left(2h \cdot c + h + N \cdot w \cdot \varepsilon\right)}{w' \cdot \left(3h' + N \cdot w' \cdot \varepsilon\right)}. \quad (7)$$

According to Equation (7), for the previous case $\eta = 13$. The resolution reduction will obviously affect the accuracy of the depth map, as analyzed in Section 6.4.

5.2.4 Enhanced memory optimizations

A different way to save memory in systems characterized by very low capabilities consists in the exploitation of a particular property of the score matrix: the score value calculation for the element located in $e_{i,j}$ needs information about the score elements located in the following three positions: $(i-1, j)$, $(i-1, j-1)$ and $(i, j-1)$. Since the score matrix building occurs by rows, for each processed row i is possible to keep in memory just information about i and $i-1$ rows: considering only the score matrix, rather than a memory occupancy of $w^2 \cdot \varepsilon$ bytes, it will require $2w \cdot \varepsilon$ bytes. The gap directions for the backtracking procedure are stored in a rewritable $w \times w$ matrix; for alignment purposes this memory unit need four bits instead of three, hence the required memory is $w^2/2$ bytes. Considering also input $(2w \cdot h \cdot c)$ and output $(w \cdot h)$ data, the required memory will be:

$$M'' = w \cdot \left(2h \cdot c + h + 2\varepsilon + \frac{w}{2}\right). \quad (8)$$

Through an enhanced memory optimization, called *line version*, the system does not store neither the whole input image pairs nor the processed disparity map, but just acquires one by one a pair of lines and returns the correspondent stereo line, independently of each other. In this case the memory requirement is:

$$M''' = w \cdot \left(2c + 1 + 2\varepsilon + \frac{w}{2}\right). \quad (9)$$

Benefits from these memory optimizations will be shown in Section 6.5.

5.2.5 Backtracking

In comparison with the original score matrix building algorithm, the maximum value searching procedure has been removed, since the highest value is always in the right-bottom corner of the score matrix, that is $e_{w,w}$.

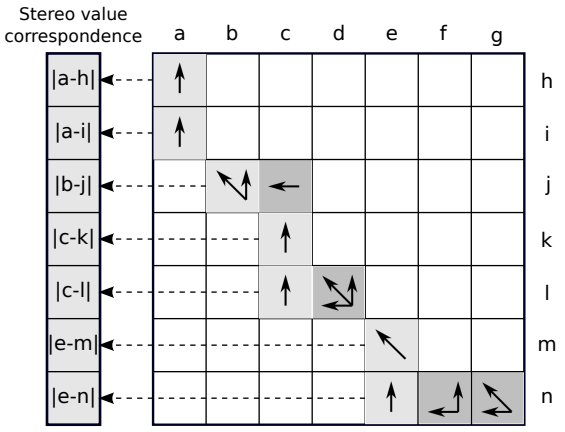


Fig. 6: Backtracking with $w = 7$ and *west - diag - north* as direction priority.

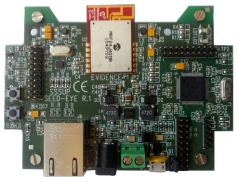
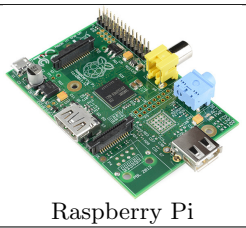
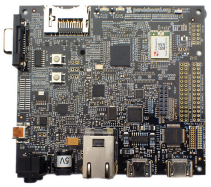
Therefore, the time and space complexity is reduced by $O(w)$. In addition, the backtrack routine has been redesigned in order to reduce processing cost and avoid recursive function calls to prevent new memory allocation and call stack management [3]. In the original algorithm, the resulting global alignment was compared with the alignments of the previous and the following line and was used to select the best of several solutions regarding the global alignment, hence reducing vertical discontinuities. In the proposed implementation the algorithm always gives priority to one direction between *north*, *west* or *diag*, depending on a manual configuration. As shown in Section 6, the results in terms of accuracy obtained by giving priority to a single direction are similar. Therefore, further processing in order to build a complete path, from the maximum score value $e_{w,w}$ to the origin score value $e_{1,1}$, has not been taken into account. In general, this simplification implies that only one possible solution is analyzed, and although the algorithm is not able to know if the best one has been selected, this has a minimum effect in the algorithm performance.

The resulting stereo value $e^S_{i,j}$ for each element of the current line i is only stored when the tracking moves towards *north* or *diag*, as shown in Figure 6. The number of steps required to reach the origin $e_{1,1}$ may vary: in case the path taken corresponds to the diagonal of the matrix, i.e. only the *diag* direction is chosen, only w steps are required. In the worst case the backtracking procedure needs $2w-1$ steps, e.g. w steps in the *west* direction and $w-1$ steps in the *north* direction.

6 Experimental results

In this section, first hardware and software setups are presented. Then, performances are evaluated in terms of quality, memory and timing by using standard images and NIR images in a night environment.

Table 1: Hardware platforms with list of features.

	 Seed-Eye	 Raspberry Pi	 PandaBoard
Clock	80 MHz	700 MHz	1000 MHz
Memory	128 KB	512 MB	1024 MB
Cache	-	128 KB (L2)	1024 KB (L2)
Instr. set	PICmicro	ARMv6	ARMv7
Storage	internal 512 KB	ext. by SD-MMC	ext. by SD-MMC
Power cons.	150mA @ 5V	700mA @ 5V	n.a.
Transceiver	802.15.4	Avb by ext HW	802.11 b/g/n
Size	92.0 x 81.0 mm	85.0 x 56.0 mm	114.3 x 101.6 mm

6.1 Hardware

The boards used in this work are shown in Table 1. Details about each board are listed below. In order to acquire images from real world, the D-Link DCS-942L camera [11] has been adopted. This pocket camera, designed for video-surveillance and security solutions, can be connected via IEEE802.11n wireless protocol or Ethernet cable, and is equipped with PIR sensors for motion detection and 4 near-infrared leds for night vision. The maximum achievable resolution is 640x480 (VGA).

Seed-Eye board [57] has been developed by Scuola Superiore Sant'Anna and Evidence s.r.l. for the Ipermob project [34] which aimed at the deployment of integrated systems based on the optimization and interoperability of the chain formed by data collection systems, both road-side and vehicular based, in a urban environment. The board is based on a PIC32MX795F512L microcontroller, capable of acting as a high power wireless sensors node. It is based on a classic 5-stage instruction pipeline (*Instruction Fetch*, *Execution*, *Memory Fetch*, *Memory Align* and *Memory Writeback*), and it is equipped with CMOS camera and GPIO which can connect many types of digital and analog sensors (e.g. light sensor, photoresistor). The power supply system has been made with an integrated DC/DC converter, which can supply the board from 5V to 38V, via battery or directly with a micro-USB plug.

Raspberry Pi [51] is a single board computer, developed by Raspberry Pi Foundation. The goal was to make a cheaper board as small as a credit card, to promote the teaching of computer science in schools. The Raspberry Pi's SoC (System on Chip) is a Broadcomm 2538 including an ARM1176JZF-S and a powerful GPU. This architecture is based on a 8-stage instruction pipeline: *1st Fetch Stage*, *2nd Fetch Stage*, *Instruction Decode*, *Register Read and Issue*, *Shifter Operation*, *ALU operation*, *Saturation* and *Memory Writeback*. Booting and storage are available on a SD-Card. In a short time Raspberry Pi has become a developed board for many fields (e.g. quadcopter controller, access point, rack computing) and several types of operating systems have been ported (e.g. RaspDebian, Arch Linux ARM, RISC OS).

The PandaBoard [49] is a cheap single board computer, based on the Texas Instruments SoC. This board, available since October 2010, has been developed for general purpose applications. The CPU is an OMAP4430 with 1 GHz ARM Cortex-A9 dual-core and 8-stage instruction pipeline, 304 MHz PowerVR SGX540 GPU, IVA3 multimedia hardware accelerator, a programmable DSP and 1 GB of DDR2 SDRAM. The operating system is stored in an SD card slot up to 32 GB.

6.2 Software

In this paper both PandaBoard and Raspberry Pi use Debian-based operating systems: Ubuntu 12.04 cross compiled for ARM OMAP and Raspbian [52] respectively. GCC compiler version is 4.6.3 and *build-essentials* Debian package is required for compilation. PIC32 compiler is based on Microchip MPLAB® X IDE [43]. While Raspberry Pi and PandaBoard can be used with any Linux OS distribution, Seed-Eye runs Erika [22], a real-time multi-tasking operating system especially designed for time-constrained embedded applications and based on a set of APIs similar to those proposed by the OSEK/VDX Consortium. Erika's priority-based scheduler allows the execution of computationally intensive processes by implementing several scheduling algorithms, e.g. Fixed Priority with preemption thresholds, Stack Resource Policy, Earliest Deadline First and Resource Reservations, which can be used to schedule tasks with real-time requirements [7].

POSIX threads library [39] for Unix has been used to manage parallel computation on Raspberry Pi and PandaBoard. Thread scheduling can be controlled by `pthread_setAffinity_np()` function, in order to allocate the tasks to each core. Since the main thread does

not need to perform further operations after burst of lines allocation (see Section 5.1.1), a FIFO (First In First Out) policy set by `pthread_setschedparam()` function has been adopted. Each of the N threads computes disparity map for the allocated burst of lines independently and synchronization occurs via `pthread_join()` function, in order to guarantee the correct termination of all the threads. By using pointer arithmetic programming technique, each thread is allowed to read/modify only the portion of memory assigned to the corresponding burst of lines (for both input and output buffer in Figure 3), therefore shared memory is not subject to inconsistency. Nevertheless, in more complicated systems, e.g. distributed applications that also need to schedule network transmissions, real-time support can be provided by SCHED_DEADLINE scheduler [17], currently available in Linux kernel v3.14.4.

Regarding time measurement, in Erika OS the primitive `GetElapsedValue()` has been used to get the number of clock ticks n between the current tick value and a previously read one. Linux OS provides the function `clock_gettime()` with nanosecond precision. Given a starting time t_1 and an ending time t_2 for a monitored sequence of instructions, elapsed time is simply computed as $\Delta t = t_2 - t_1$, or $\Delta t = \frac{n}{f}$ for Erika OS with clock rate f. In case of multi-core processing, time for thread allocation has also been taken into account.

6.3 Evaluation setup

The validation of this version of the algorithm proposed by [14] is performed by using the benchmarking framework from Middlebury [56], which has been largely accepted by the computer vision community for objective comparison of stereo matching algorithms. A way to estimate the quality of the computed correspondences is to evaluate error statistics with respect to ground truth data provided by the framework.

Regarding the evaluation of NIR images, a dataset has been acquired as explained in 6.3.3. Images from infrared cameras tend to have a single color channel, since the sensor generally does not distinguish different wavelengths of infrared radiation [18]. Wavelengths outside of the visible spectrum do not map uniformly into the system of color vision used by humans. Changes in the signal can be displayed from monochromatic or pseudo-colors images using changes in color rather than in intensity, by analyzing lighter color regions[4] in the disparity map and dissociating them from the background, depending on cameras height, cameras field of view, lenses distance, and object distance [16] and [4].

[4] According to the grayscale-space representation, let assume that value 0 is black and the maximum value, i.e. 255 at 8 bits per pixel, is white. Therefore, higher values in the disparity map correspond to lighter areas in the image, i.e. objects closer to cameras.

6.3.1 Quality measures

Considering the computed disparity map $d_C(x, y)$ and a ground truth map $d_T(x, y)$ characterized by a total number of pixels N_p, the percentage of bad matching pixels B is shown in Equation (10):

$$B = \frac{1}{N_p} \sum_{(x,y)} \big(|d_C(x, y) - d_T(x, y)| > \delta_d\big). \tag{10}$$

In the previous equation δ_d is a disparity error tolerance defined by the Middlebury's framework and set to default value $\delta_d = 1.0$.

In addition to this overall metric, statistics related to specific regions of the disparity maps were also considered in order to support the analysis of matching results in typical problem areas. In particular, *occluded regions* $\mathcal{O}$ represent occluded areas in the matching image, i.e., where the forward-mapped disparity lands at a location with a larger or nearer disparity. These regions are computed by pre-processing both reference image and ground truth disparity map; the resulting metric $B_{\overline{\mathcal{O}}}$ represents the percentage of bad matching pixels in non occluded areas.

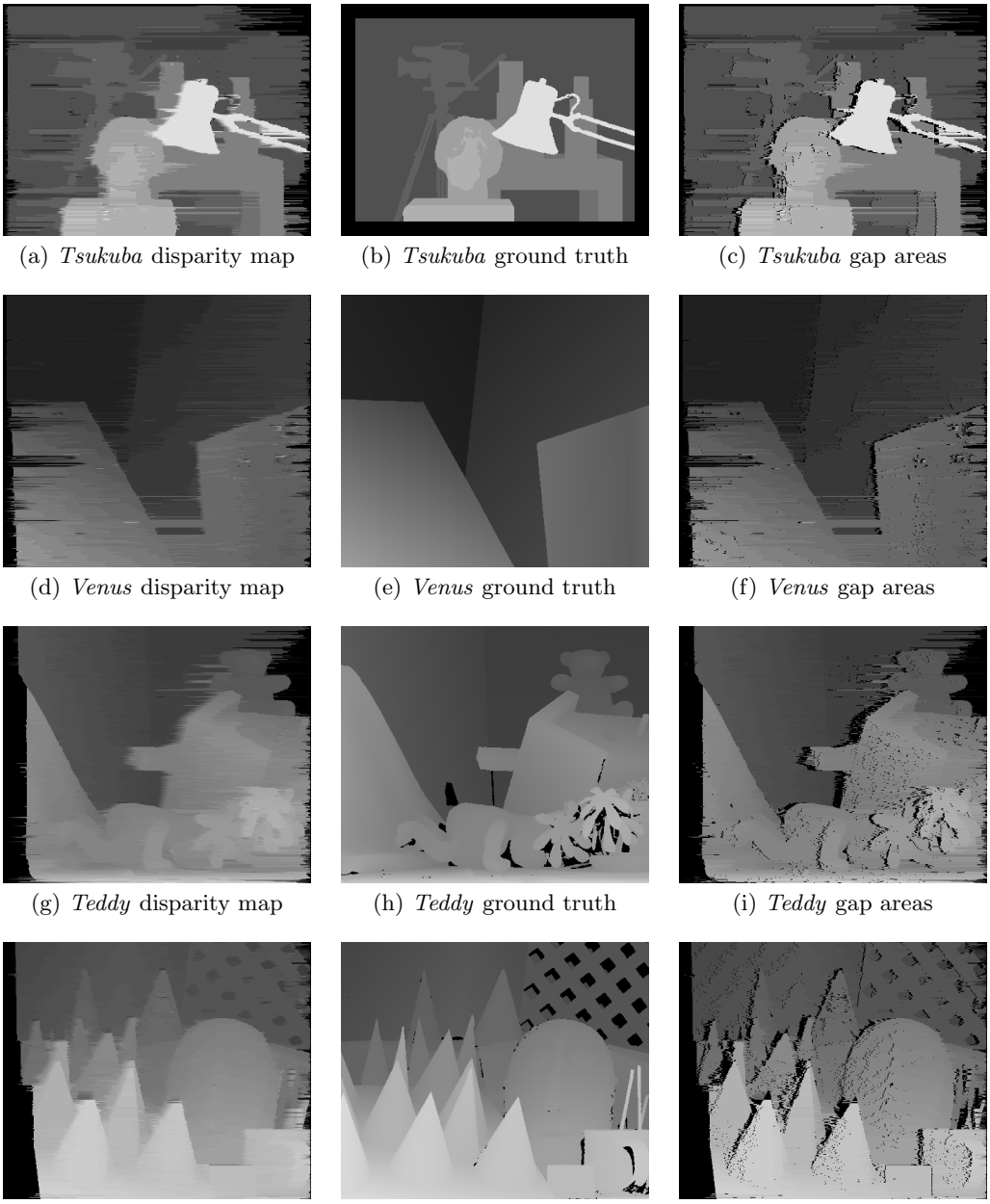
6.3.2 Parameter configuration

In [14] score parameters, as *match*, *gap* and *extended gap*, could be dynamically configured by generating a pseudo ground truth disparity map and establishing sparse pixel correspondences using the SIFT (Scale-Invariant Feature Transform) algorithm. Since these parameters are sensitive to image characteristics, they need to be customized for each stereo pair. If the actual disparity map is known, an optimization function can be applied to maximize matching accuracy. Although such process allows evaluating the best possible performance of an algorithm, it does not have practical applications.

In order to make the processing as fast as possible and to compare in a fair way all the image pairs, the same set of parameters, according to the default values in the previous work [14], are used. Without any *a priori* knowledge about the analyzed images and no automatic parameter configuration enabled, let assume for gap and extended gap parameters the following values: $gap = 69$ and $egap = 98$.

6.3.3 NIR dataset

In order to demonstrate the effectiveness of the proposed work, even in case of an input data stream, a scenario aiming at detecting objects or pedestrians passing through a gateway is proposed. Two cameras are installed on top of a gateway (indoor setting), facing the ground, counting the number of entities and estimating their height. The camera pair is at a short distance from



(a) *Tsukuba* disparity map (b) *Tsukuba* ground truth (c) *Tsukuba* gap areas

(d) *Venus* disparity map (e) *Venus* ground truth (f) *Venus* gap areas

(g) *Teddy* disparity map (h) *Teddy* ground truth (i) *Teddy* gap areas

(j) *Cones* disparity map (k) *Cones* ground truth (l) *Cones* gap areas

Fig. 7: Computed disparity maps, ground truth and gap areas insertion comparison.

each other and use the same base line. Under these assumptions rectification constraints can be relaxed. The resulting disparity map has been analyzed by varying brightness conditions, the input image pairs colormap (RGB, grayscale, NIR) and also the target distance, by changing its position from standing to kneeling. The NIR images have been taken under conditions of complete darkness.

6.4 Stereo quality performances

In the following stereo quality performances for Middlebury's dataset and proposed NIR dataset are presented.

6.4.1 Middlebury's disparity map analysis

Figure 7 shows a comparison between computed disparity maps and ground truth disparity maps obtained for the images used in the Middlebury framework. As mentioned in [56], the use of the same set of parameters for

Table 2: Stereo algorithm results, where $B_{\overline{\mathcal{O}}}$ is the percentage of bad matching pixels in non occluded areas, B is the percentage of bad matching pixels in the whole computed disparity map.

Algorithm	Tsukuba 384 × 288		Venus 434 × 383		Teddy 450 × 375		Cones 450 × 375		Average bad pixels
	$B_{\overline{\mathcal{O}}}$	B	$B_{\overline{\mathcal{O}}}$	B	$B_{\overline{\mathcal{O}}}$	B	$B_{\overline{\mathcal{O}}}$	B	%
Dynamic Programming [5]	4.12	5.04	10.1	11.0	14.0	21.6	10.5	19.1	**14.2**
Previous work [14]	6.74	8.91	10.7	12.2	14.1	23.0	11.0	21.2	**16.7**
Our approach (RGB, *west* priority)	5.80	7.87	9.76	10.9	13.9	20.9	9.60	16.7	**15.3**
Our approach (RGB, *north* priority)	5.79	7.82	9.81	10.9	13.9	20.9	9.68	16.8	**15.3**
Our approach (RGB, *diag* priority)	5.77	7.83	9.84	11.0	14.0	21.0	9.73	16.8	**15.4**
Our approach (RGB, half res, *diag* priority)	6.72	8.80	7.04	8.14	11.25	16.29	8.40	15.56	**15.3**
Our approach (Gray, *diag* priority)	10.27	12.24	13.74	14.82	24.41	30.96	19.04	25.43	**23.5**
Our approach (Gray, half res, *diag* priority)	10.31	12.26	11.60	12.72	19.64	25.65	25.79	19.20	**22.9**

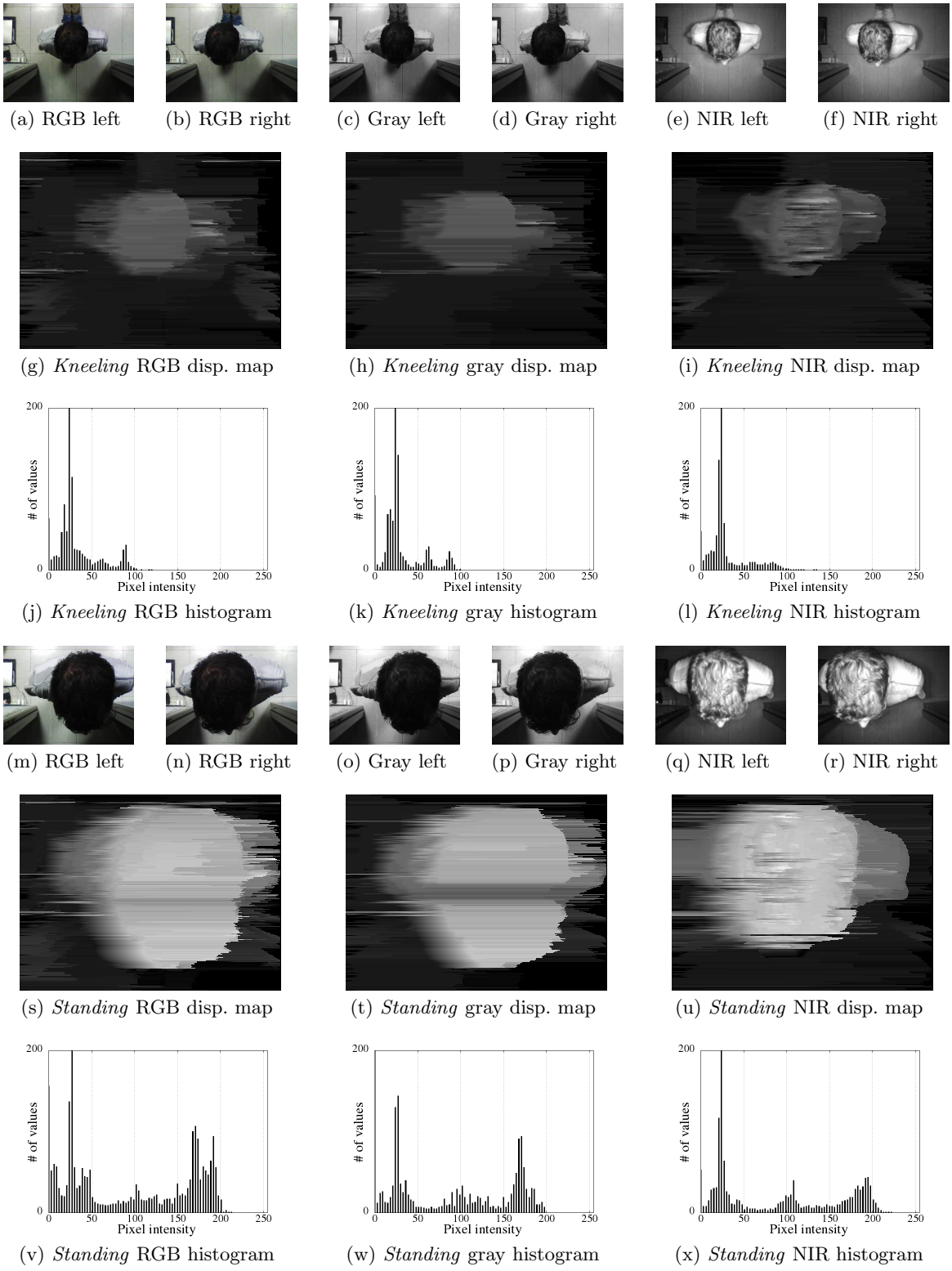
all four datasets is required. Some image pairs are noisier than the others, which works against algorithms that are sensitive to internal parameter settings.

The proposed line-by-line approach is particularly evident in Venus image pair, due to the presence of objects with well-defined edges. The disparity maps created by scanline based algorithms are fairly detailed, but the larger qualitative errors are clearly a result of the artifacts due to the lack of inter-scanline consistency, specially near discontinuities. Given two lines l_{h^*} and r_{h^*} located at height h^* of the input image pairs I_{left} and I_{right} with resolution $w \times h$, the original algorithm did not compute disparity estimates in low confidence regions. Those unmatched areas explicitly model occlusions, and were filled with black color [14]. As explained in Section 5.2.5, in the proposed work the unmatched areas have been filled according to the following score direction policy: given a stereo output line s_{h^*}, the value of the j-th element $s_{h^*}[j]$ is actually stored only if the next processed element in the score matrix E will be at row $j-1$, i.e. border of the unmatched area is reached. The qualitative differences between these two techniques can be noticed by comparing the first and third column in Figure 7. Since the average bad pixels results obtained by the evaluation framework for the new stereo line building policy were between 1% and 2% better, the new strategy was adopted. Moreover, in the case of an application for object detection, gap areas could cause several issues on the histograms analysis in the disparity maps based on color intensity, as explained in Section 6.3.

In Table 2, quantitative results are provided to validate the proposed algorithm. The resolution for Tsukuba, Venus, Teddy and Cones image pair is respectively 384×288, 434×383, 450×375 and 450×375. In order to compare in a fair way the results of the previous work [14], the extended gap (EG) capability has been taken into account, without considering median filter (MF) and automatic parameter selection (AP) procedures. The application of a median filter on disparity maps introduces some inter-scanline coherence, increasing also accuracy, but this operation involves scanning of pixels in each row and in each column, with an additional time complexity $O(w \cdot h)$. Performances for a standard method are also provided: the reference for scanline-based Dynamic Programming (DP) [5] optimizes parameters by using a particular data structure called *disparity-space image*, i.e. an explicit spatial representation of matching points for each line l and r of the input image pair.

Obviously, in terms of accuracy, the proposed approach cannot compete with more computationally expensive approaches, but the results are satisfactory. With respect to the previous work, this approach shows fewer bad matching pixel for all the image pairs and for all the proposed prioritized backtracking directions: at least -0.9% bad pixels for Tsukuba, -1.2% for Venus, -2.0% for Teddy, -4.4% for Cones image pair. The average bad pixel error is 1.4% better than in the previous work. The difference between the direction priorities is negligible (for the previous image pairs is $\pm 0.05\%$, $\pm 0.01\%$, $\pm 0.01\%$, $\pm 0.01\%$ respectively), since the average bad pixel metric does not show any significant difference.

In the lower part of the table quality results about color channel and image resolution reduction are shown. For the sake of simplicity, only results regarding *diag* direction priority have been reported. The length and width halving in input image pairs does not affect matching quality. Small variations in overall bad matching pixel for Middlebury's dataset can be explained by considering that both ground truth and computed disparity map had to be shrunken, hence some bad pixel at previous resolu-



(a) RGB left (b) RGB right (c) Gray left (d) Gray right (e) NIR left (f) NIR right

(g) *Kneeling* RGB disp. map (h) *Kneeling* gray disp. map (i) *Kneeling* NIR disp. map

(j) *Kneeling* RGB histogram (k) *Kneeling* gray histogram (l) *Kneeling* NIR histogram

(m) RGB left (n) RGB right (o) Gray left (p) Gray right (q) NIR left (r) NIR right

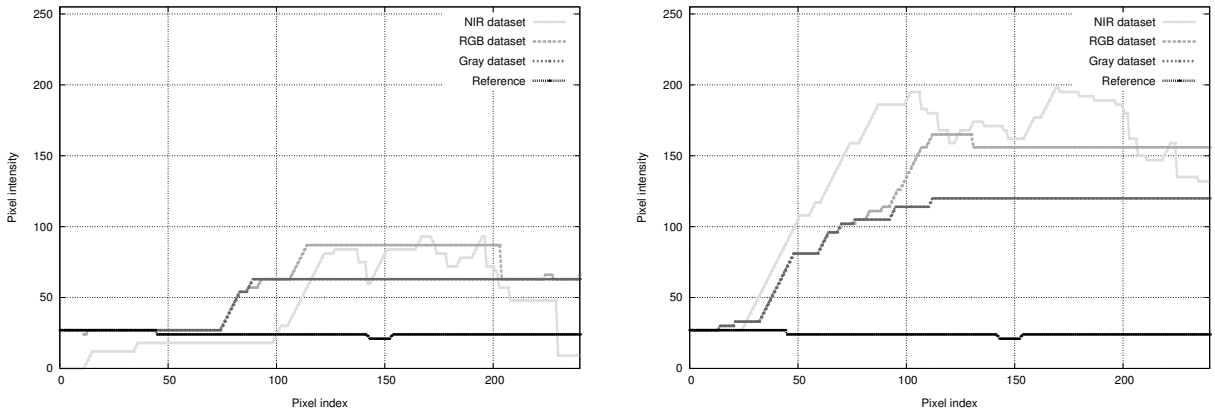
(s) *Standing* RGB disp. map (t) *Standing* gray disp. map (u) *Standing* NIR disp. map

(v) *Standing* RGB histogram (w) *Standing* gray histogram (x) *Standing* NIR histogram

Fig. 8: Input image pairs, disparity maps and histograms for different positions.

Table 3: Components for the Gaussian Mixture Model.

# Component	Kneeling position						Standing position					
	RGB		Gray		NIR		RGB		Gray		NIR	
	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2	μ	σ^2
g_1	25	18	25	18	24	17	26	27	27	19	26	18
g_2	62	17	63	11	76	52	100	81	100	85	104	20
g_3	92	8	92	11	-	-	173	51	171	34	188	37



(a) Kneeling position (b) Standing position

Fig. 9: Spatial distribution for a single disparity map line $\overline{h} = h/2$.

tion, especially close to discontinuities, simply could disappear at new resolution. Color channel transformation from RGB to grayscale at full resolution shows significant worsening in the stereo quality: +4.41%, +3.82%, +9.96% and +8.63% respectively. The larger number of bad pixels for Teddy and Cones image pairs can be explained by the higher range of disparity values, i.e. 60, compared to Tsukuba and Venus, i.e. 16 and 20 respectively [56].

However, the reduction in the number of channels causes a degradation in the quality of the resulting disparity map around 7-8%. Considering both the effects of resolution and channel reduction, respect to the full size RGB case, bad matching pixels in the whole disparity map for Middlebury's dataset increases by the following ΔB values: $\Delta B^{tsukuba} = +4.43\%$, $\Delta B^{venus} = +1.72\%$, $\Delta B^{teddy} = +4.65\%$, $\Delta B^{cones} = +2.40\%$. Therefore, the resulting data reduction causes a degradation in the quality of the stereo results lower than 5%. The average value $\mu_{\Delta B}$ for the previous bad matching pixel percentages is:

$$\mu_{\Delta B} = +3.3\%. \tag{11}$$

Such a small value for the average bad matching pixel percentage allows the use of the only luminance channel in the input image pairs at half resolution in order to build the disparity map. This results in a timing performance improvement and a decrease of required memory, as explained in the next section.

6.4.2 NIR disparity map analysis

In Figure 8 results assessing the applicability of the proposed embedded version, in both day and night environments, are shown. The NIR disparity map is robust to illumination changes and requires lower memory and timing resources compared to the RGB one.

The resulting disparity maps by changing input image pairs conditions (Figure 8g, 8h and 8i for kneeling position, Figure 8s, 8t and 8u for standing position) show a similar qualitative behavior for a given position, while corresponding histograms (Figure 8j-8x) show different pixel intensity distributions depending on the position. Even in the near-infrared, disparity map for the standing position highlights distinct pixel intensity areas corresponding to background, shoulders and head. Approximation due to the color-space conversion in the input image pairs could cause the occurrence of some artifacts, as shown in Figure 8t. In this case, for some lines in the original input image, two different colors have been mapped in the same gray level, jeopardizing the recognition of areas at different distances from the camera.

By analyzing the resulting histograms, the system is able to detect, even in different color spaces, if the human target is in a kneeling or standing position. The two positions are characterized by different mixtures of Gaussians. Table 3 reports the Gaussian parameters (μ, σ^2) for three components g_1, g_2, g_3 of the mixture. The first component g_1 represents the background. In case of both

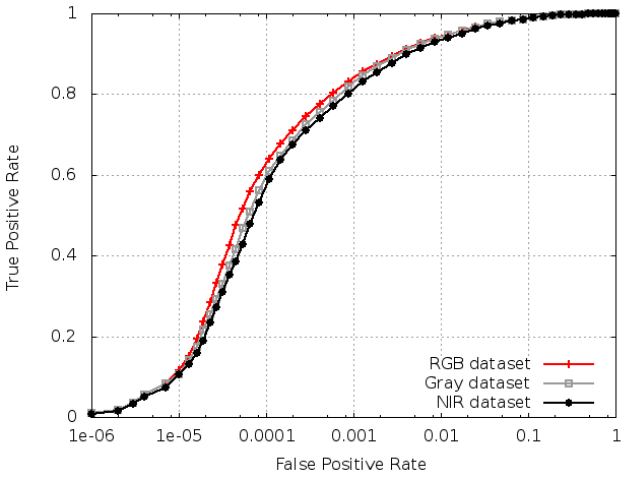


Fig. 10: ROC curves for the proposed application.

kneeling and standing position the mean value is always in the range $[25, 27]$, despite the different color space. The second component g_2 represents the target's shoulders. In case of kneeling position, the mean value is in the range $[62, 63]$ for RGB and grayscale color space, while for NIR color space g_2 and g_3 are not separable, as shown in Figure 8i, since colors appear quite mixed. For the standing position, μ_{g_2} is in the range $[100, 104]$. The third component g_3 represents the target's head and the mean value is equal to 92 for both RGB and grayscale images in the kneeling position. For the standing position, mean values are 173, 171 and 188 respectively. The NIR component comes out slightly shifted, but the value is still comparable to the previous ones.

The spatial distribution of the gray-levels intensity for a single horizontal line located at $\overline{h} = h/2$ in the disparity map is depicted in Figure 9. This line should be able to detect all the objects passing through the gateway. The graphs compare the resulting line respect to RGB, grayscale and NIR input image pairs and a reference case without any object in the scene. According to the different color spaces, the line representing a kneeling position produces pixel intensity values for the head area in the range $[25, 80]$, while line representing a standing position produces values in the range $[120, 200]$. It should be noted that the line trend in NIR case does not look as linear as the other cases, due to the infrared illuminator that could introduce some light reflections depending on the type of surface.

Given the histogram results in Figure 8, a color transformation can be applied to the disparity maps in order to obtain uniform areas for objects approximately at the same distance from the stereo system. 1000 frames per color space have been used to set the best thresholds to separate the components g_1, g_2 and g_3 in case of kneeling (th_{12}^K and th_{23}^K) or standing position (th_{12}^S and th_{23}^S). After color transformation, difference between the size of uniform areas is used as threshold to recognize the position. The Figure above shows the Receiver Operating Characteristic curves for the proposed application by varying $\left(th_{12}^K,\, th_{23}^K,\, th_{12}^S,\, th_{23}^S\right)$. For each frame, ground truth has been manually annotated and consists of one of these states: *a)* no target, *b)* target in a kneeling position and *c)* target in a standing position. Considering false positive rate (FPR) values in the interval $[10^{-6}, 1]$ and true positive rate values for RGB dataset $\left(TPR^{RGB}\right)$, grayscale dataset $\left(TPR^{Gray}\right)$ and NIR dataset $\left(TPR^{NIR}\right)$, resulting curves are very similar, e.g. $TPR^{RGB} - TPR^{NIR} < 0.1$ at $FPR = 10^{-4}$ and $TPR^{RGB} - TPR^{NIR} \approx 0.05$ at $FPR = 10^{-3}$. Consequently, color space reduction even in case of NIR input images does not affect the suitability of the application.

6.5 Timing performances

Table 4 lists a summary of all the processing time measurements taken on the available boards. Optimizations explained in Section 5.2.2 have been taken into account only for PIC32 board. Every image pair with $w \times h$ resolution has been processed both in a full-size and half-size mode according to the memory requirements explained in Section 5.2.3. The overall processing time to get a stereo image (h lines) is proportional to the single-line score matrix building procedure ($w \cdot w$ elements). In case of full-size images the expected overall processing time t_{full} can be represented by the following:

$$t_{full} \approx h \cdot w^2. \tag{12}$$

In case of half-size images, the processing time t_{half} can be introduced:

$$t_{half} \approx \frac{h}{2} \cdot \frac{w}{2} \cdot \frac{w}{2} = \frac{1}{8}\, h \cdot w^2. \tag{13}$$

Results listed on the table confirms the validity of the factor $\tau_{res} = t_{full}/t_{half} = 8$ in the execution time of the algorithm between images at different resolutions: e.g. $\tau_{res} = 7.92$ for Cones dataset with 3 channels on Raspberry board, $\tau_{res} = 7.63$ for Tsukuba dataset with 1 channel on dual-core PandaBoard, and $\tau_{res} = 7.94$ for Venus dataset with 3 channels on PIC32 board.

The factor τ_{ch} represents the ratio between the processing time for images with 3 color channels and the processing time for images with a single color channel. Different boards are characterized by different values of this metric: $\tau_{ch} \approx 1.5$ for dual-core PandaBoard, $\tau_{ch} \approx 2.8$ for Raspberry board, $\tau_{ch} \approx 1.3$ for PIC32 board. Time saved deals with the procedure to compute the mismatch value Δ_{ij}^{diag} for each cell in the score matrix, according to Eq. (1). Images with 3 channels need the sum of three absolute differences of pixel values, while in case of single channel only one absolute difference is required.

Table 4: Timing results.

Input		Intel Quad-Core 2.66 GHz			PandaBoard 1.0 GHz		Rasp. Pi 0.7 GHz	Microchip PIC32 80 MHz	
Dataset	Resolution	4 cores (s)	2 cores (s)	1 core (s)	2 cores (s)	1 core (s)	1 core (s)	v. full[1] (s)	v. line[2] (s)
Tsukuba	full, 3 ch.	0.411	0.707	1.401	4.498	8.917	37.469	-	65.266
	full, 1 ch.	0.188	0.371	0.736	2.982	5.849	18.563	-	49.585
	half, 3 ch.	0.079	0.103	0.205	0.573	1.139	5.481	-	8.293
	half, 1 ch.	0.024	0.047	0.093	0.391	0.770	3.184	6.230	6.320
Venus	full, 3 ch.	0.750	1.004	1.969	7.515	14.709	59.611	-	112.661
	full, 1 ch.	0.324	0.625	1.236	4.813	9.357	22.785	-	86.088
	half, 3 ch.	0.106	0.135	0.265	0.955	1.884	8.669	-	14.192
	half, 1 ch.	0.042	0.078	0.155	0.623	1.232	4.101	-	10.858
Teddy	full, 3 ch.	0.729	1.052	2.093	7.664	15.099	56.839	-	118.434
	full, 1 ch.	0.339	0.657	1.302	4.962	9.637	20.209	-	90.521
	half, 3 ch.	0.102	0.133	0.261	0.948	1.884	7.172	-	14.861
	half, 1 ch.	0.043	0.084	0.164	0.624	1.224	2.549	-	11.395
Cones	full, 3 ch.	0.812	1.084	2.130	7.716	15.201	56.815	-	118.434
	full, 1 ch.	0.348	0.672	1.328	5.067	9.848	20.279	-	90.521
	half, 3 ch.	0.106	0.135	0.268	0.955	1.901	7.173	-	14.860
	half, 1 ch.	0.043	0.084	0.168	0.627	1.249	2.553	-	11.395

[1] *Full* version: both input and output images stored in the memory of the board.
[2] *Line* version: the board receives input lines and returns the stereo line.

An Intel® Core™ 2 Quad Processors [33] has been taken as a reference system to compare all the processing time results. The table shows results by taking into account one, two and all four cores. The single core Raspberry Pi board takes almost one minute for full-size Cones, Venus and Teddy datasets, and 37 *s* for Tsukuba. In case of half-size resolution and 1 channel, results are between 2.5 *s* and 4.1 *s*. Obviously this board could be used only in very low-rate applications with static objects, such as parking monitoring.

The Seed-Eye board needs the special memory modification explained in Section 5.2.4 in order to make the implementation feasible. As depicted in Table 5, in case of QQVGA input images with 1 or 3 channels, the execution of the optimized version of the proposed algorithm does require more than 128 KB of memory. Furthermore, in order to develop a meaningful application onboard and execute some kind of processing, i.e. detection or classification algorithms, more memory is required. The processing time required by the memory optimization method shown in Eq. (8) is between 0.8% and 1.4% more than the previous approach, but in case of QQVGA resolution images, the required memory decreases by 46.9%. As shown in Table 4, the PIC32 board is able to execute the stereo algorithm only for Tsukuba half-size images, with a single channel, in 6.230 *s*.

Last memory optimization shown in Table 5 has been taken into account to make an overall comparison between the Middlebury's image pairs for timing requirements also on the PIC32 board. By using the *line version* of memory optimization presented in Eq. (9), the required memory is very low: 209 KB for 3-channels VGA images, and only 14 KB for 1-channel QQVGA images. By focusing just on half resolution 1-channel images, timing results for Tsukuba, Venus, Teddy and Cones are 6.320 *s*, 10.858 *s*, 11.395 *s* and 11.395 *s* respectively.

Table 5: Memory requirements.

Resolution	Ch.	No opt. version[1] (KB)	Opt. version[2] (KB)	Full version[3] (KB)	Line version[4] (KB)
VGA	3	4900	3300	2304	209
640 × 480	1	3700	2100	1104	206
QVGA	3	1125	825	577	55
320 × 240	1	925	525	277	53
QQVGA	3	307	207	145	15
160 × 120	1	232	132	70	14

[1] *No opt. version*: no memory optimizations.
[2] *Opt. version*: memory optimizations according to Eq. (6).
[3] *Full version*: memory optimizations according to Eq. (8).
[4] *Line version*: memory optimizations according to Eq. (9).

The impact of serial transmission on overall timing performance is relevant. For instance, input image pair transmission and disparity map reception for Tsukuba dataset takes 53.76 *s* in case of 3-channels full resolution, and 5.76 *s* in case of 1-channel half resolution. For Cones dataset, the same experiment results in 82.03 *s* and 8.79 *s* respectively. PIC32 microcontroller cannot obviously be used as coprocessor, since serial communication is a significant bottleneck, but it is a useful support in systems such as low-cost vehicle counter [54].

In case of half-size images, the dual-core PandaBoard always evaluates stereo image pairs in less than 1 second. Figure 11 summarizes the distribution of bad matching pixels with respect to computing time in PandaBoard with multi-core processing enabled. Similar graphs can be obtained with different boards. Quality results for low resolution dataset images are fairly similar to the

Table 6: Timing performance comparison between different algorithm. Saved time in parentheses.

Dataset	Res.	# cores	Proposed implementation	[14] implementation	[5] implementation
Tsukuba	full	1	1.401	1.989 (-42%)	1.694 (-21%)
Tsukuba	half	1	0.205	0.328 (-60%)	0.245 (-20%)
Tsukuba	full	2	0.707	1.001 (-42%)	0.862 (-22%)
Tsukuba	half	2	0.103	0.141 (-37%)	0.113 (-10%)
Venus	full	1	1.969	2.813 (-43%)	2.772 (-41%)
Venus	half	1	0.265	0.466 (-76%)	0.358 (-35%)
Venus	full	2	1.004	1.426 (-42%)	1.376 (-37%)
Venus	half	2	0.135	0.258 (-91%)	0.176 (-30%)
Teddy	full	1	2.093	2.976 (-42%)	2.664 (-27%)
Teddy	half	1	0.261	0.473 (-81%)	0.333 (-27%)
Teddy	full	2	1.052	1.511 (-44%)	1.387 (-32%)
Teddy	half	2	0.133	0.169 (-27%)	0.164 (-23%)
Cones	full	1	2.130	3.140 (-47%)	2.713 (-27%)
Cones	half	1	0.268	0.392 (-46%)	0.323 (-21%)
Cones	full	2	1.084	1.508 (-39%)	1.355 (-25%)
Cones	half	2	0.135	0.208 (-54%)	0.171 (-26%)

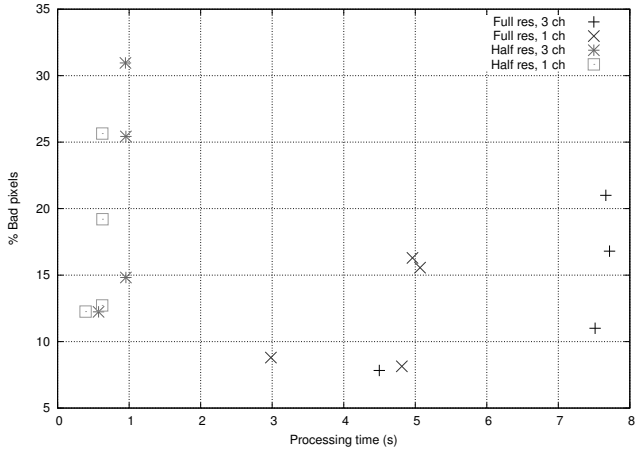


Fig. 11: Processing time w.r.t. bad matching pixels distribution run on multi-core PandaBoard. A group of 4 similar points represents the group of 4 image pairs from Middlebury's dataset at a given resolution.

ones at higher resolution, but the benefits are related to the faster way to obtain the disparity map. In the graph the process for Tsukuba grayscale image pairs at half resolution 192×144 takes 0.391 s. This resolution is fairly close to QQVGA (160×120). Therefore, by scaling the dataset to QQVGA resolution, the process can be completed even faster: Tsukuba 0.229 s, Venus 0.219 s, Teddy 0.229 s and Cones 0.231 s. These results allow the processing of stereo images, with the low-cost board above mentioned and the proposed algorithm, at a rate of 4 frames per second.

In order to assess the timing performance improvement respect to previous works, Table 6 shows timing results for the Intel platform by implementing the algorithms described in [14] and [5] with parallel computation enabled. For the sake of simplicity only 3-channels images have been considered. The excellent results obtained with respect to [14] are mainly due to the simplified backtracking procedure which avoids recursive functions for multiple paths analysis involving neighboring stereo lines. Regarding the Dynamic Programming technique in [5], only occlusion and ordering constraints have been evaluated. By means of the proposed implementation, average time saved is greater than 50% for [14] and greater than 25% for [5].

7 Conclusions

In this paper, the applicability of a computationally demanding stereo matching algorithm in different low-cost and low-complexity embedded devices has been explored, by focusing on the analysis of timing and image quality performances. Reduction of color channel information and resolution for input images can decrease the amount of required memory up to a factor of 25, while low-level software optimizations and analysis of redundant data structures and internal data representation do not affect the effectiveness of the stereovision system. The proposed implementation performs well in spite of the optimizations even in a night environment by means of infrared sensors. Hence, regardless of lighting conditions, objects placed at different distances can be easily detected.

Low cost boards Raspberry and PIC32 can be used in applications which do not need service rate higher than few disparity maps per minute. A typical application could be the parking monitoring for empty stalls detection. PandaBoard can be successfully used to obtain disparity maps at QQVGA resolution, by using the proposed stereo matching technique, at a rate of 4 fps.

Future work will consider the development of an algorithm able to detect in real-time the presence and the distance of a moving target in the stereo cameras range

by considering a 3D scoring matrix extension [41] of the proposed stereo matching implementation.

References

1. Ambrosch K., Kubinger W. (2010) Accurate hardware-based stereo vision. Computer Vision and Image Understanding 114(11):1303 – 1316, DOI http://dx.doi.org/10.1016/j.cviu.2010.07.008
2. Baker H.H., Binford T.O. (1981) Depth from Edge and Intensity Based Stereo. In: Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, IJCAI'81, pp 631–636.
3. Banahan M., Brady D., Doran M. (1991) The C Book: Featuring the ANSI C Standard. Addison-Wesley.
4. Bertozzi M., Broggi A., Caraffi C., Rose M.D., Felisa M., Vezzoni G. (2007) Pedestrian detection by means of far-infrared stereo vision. Computer Vision and Image Understanding 106(23):194 – 204, DOI http://dx.doi.org/10.1016/j.cviu.2006.07.016
5. Bobick A.F., Intille S.S. (1999) Large Occlusion Stereo. International Journal of Computer Vision 33(3):181–200, DOI 10.1023/A:1008150329890
6. Boykov Y., Veksler O., Zabih R. (2001) Fast approximate energy minimization via graph cuts. Pattern Analysis and Machine Intelligence, IEEE Transactions on 23(11):1222–1239, DOI 10.1109/34.969114
7. Buttazzo G.C. (2004) Hard Real-time Computing Systems: Predictable Scheduling Algorithms And Applications (Real-Time Systems Series). Springer-Verlag TELOS, Santa Clara, CA, USA.
8. Chandrapala T. (2011) Real time Stereo Vision based on biologically motivated algorithms using GPU. 17th ERU Research Symposium: University of Moratuwa, Sri Lanka.
9. Chen D., Varodayan D., Flierl M., Girod B. (2008) Distributed stereo image coding with improved disparity and noise estimation. In International Conference on Acoustics, Speech and Signal Processing (ICASSP).
10. Cox I.J., Hingorani S.L., Maggs B.M., Rao S.B. (1996) A Maximum Likelihood Stereo Algorithm. Computer Vision and Image Understanding 63(3):542–567.
11. D-Link (2011) DCS-942L Datasheet. http://www.dlink.com/.
12. Deng Y., Lin X. (2006) A Fast Line Segment Based Dense Stereo Algorithm Using Tree Dynamic Programming. In: Leonardis A., Bischof H., Pinz A. (eds) Computer Vision ECCV 2006, Lecture Notes in Computer Science, vol 3953, Springer Berlin Heidelberg, pp 201–212, DOI 10.1007/11744078_16
13. Devy M., Boizard J.L., Galeano D.B., Lindado H.C., Manzano M.I., Irki Z., Naoulou A., Lacroix P., Fillatreau P., Fourniols J.Y., Parra C. (2011) Stereovision Algorithm to be Executed at 100Hz on a FPGA-based Architecture. In: Advances in Theory and Applications of Stereo Vision, Dr. Asim Bhatti, DOI 10.5772/14037
14. Dieny R., Thevenon J., Martinez-del-Rincon J., Nebel J.C. (2011) Bioinformatics inspired algorithm for stereo correspondence. In: VISAPP, Vilamoura, Portugal.
15. Dos Santos-Paulino A., Nebel J.C., Florez-Revuelta F. Evolutionary algorithm for dense pixel matching in presence of distortions, EvoStar, Granada, Spain, 23-25 April 2014.
16. Dubbelman G., Van der Mark W., Van Den Heuvel J.C., Groen F.C.A. (2007) Obstacle detection during day and night conditions using stereo vision. In: Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on, pp 109–116, DOI 10.1109/IROS.2007.4399055
17. Faggioli D., Checconi F., Trimarchi M., Scordino C. An EDF scheduling class for the Linux kernel. 11th Real-Time Linux Workshop (RTLWS), Dresden, Germany, September 2009.
18. FLIR (2011) Thermal imaging guidebook for industrial applications.
19. Forstmann S., Kanou Y., Ohya J., Thuering S., Schmitt A. (2004) Real-Time Stereo by using Dynamic Programming. In: Computer Vision and Pattern Recognition Workshop, 2004. CVPRW '04. Conference on. pp 29–29, DOI 10.1109/CVPR.2004.154
20. Forsyth D.A., Ponce J. (2002) Computer Vision: A Modern Approach. Prentice Hall Professional Technical Reference.
21. Fowers J., Brown G., Cooke P., Stitt G. (2012) A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications. In Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays (FPGA '12). ACM, New York, NY, USA, 47-56.
22. Gai P., Bini E., Lipari G., Natale M.D., Abeni L. (2000) Architecture for a Portable Open Source Real Time Kernel Environment. In: Proceedings of the Second Real-Time Linux Workshop and Hand's on Real-Time Linux Tutorial.
23. Gaisler J. (2002) A Portable and Fault-Tolerant Microprocessor Based on the SPARC V8 Architecture. In Proceedings of the 2002 International Conference on Dependable Systems and Networks (DSN '02). IEEE Computer Society, Washington, DC, USA, 409-415.
24. Geiger D., Ladendorf B., Yuille A. (1995) Occlusions and binocular stereo. International Journal of Computer Vision 14(3):211–226, DOI 10.1007/BF01679683

25. Goss C.F. (2013) Machine Code Optimization - Improving Executable Object Code. CoRR abs/1308.4815.
26. Gudis E., Van der Wal G., Kuthirummal S., Chai S. (2012) Multi-Resolution Real-Time Dense Stereo Vision Processing in FPGA. In: Field-Programmable Custom Computing Machines (FCCM), 2012 IEEE 20th Annual International Symposium on. pp 29–32, DOI 10.1109/FCCM.2012.15
27. Held M., Karp R.M. (1961) A Dynamic Programming Approach to Sequencing Problems. In: Proceedings of the 1961 16th ACM National Meeting (ACM '61), New York, NY, USA, pp 71.201–71.204, DOI 10.1145/800029.808532
28. Hengstler S., Aghajan H. (2006) A Smart Camera Mote Architecture for Distributed Intelligent Surveillance. In ACM SenSys Workshop on Distributed Smart Cameras (DSC).
29. Higgins D., Thompson J., Gibson T., Thompson J.D., Higgins D.G., Gibson T.J. (1994) Improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Research 22:4673–4680.
30. Hirschmuller H., Scharstein D. (2007) Evaluation of Cost Functions for Stereo Matching. In: Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on. pp 1–8, DOI 10.1109/CVPR.2007.383248
31. Humenberger M., Zinner C., Weber M., Kubinger W., Vincze M. (2010) A fast stereo matching algorithm suitable for embedded real-time systems. Computer Vision and Image Understanding 114(11):1180 – 1202, DOI http://dx.doi.org/10.1016/j.cviu.2010.03.012
32. Free Software Foundation Inc. (2013) GCC, the GNU Compiler Collection. http://gcc.gnu.org/.
33. Intel (2007) Intel® core™quad processors specifications. http://www.intel.com.
34. The IPERMOB project (2009) A Pervasive and Heterogeneous Infrastructure to control Urban Mobility in Real-Time. http://www.ipermob.org.
35. Jin S., Cho J., Pham X.D., Lee K.M., Park S.K., Kim M., Jeon J.W. (2010) FPGA Design and Implementation of a Real-Time Stereo Vision System. Circuits and Systems for Video Technology, IEEE Transactions on 20(1):15–26, DOI 10.1109/TCSVT.2009.2026831
36. Lassmann T., Sonnhammer E.L.L. (2005) Kalign - an accurate and fast multiple sequence alignment algorithm. BMC Bioinformatics 6:298.
37. Lempitsky V., Rother C., Blake A. (2007) LogCut - Efficient Graph Cut Optimization for Markov Random Fields. In: Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on. pp 1–8, DOI 10.1109/ICCV.2007.4408907
38. Levitin A.V. (2002) Introduction to the Design and Analysis of Algorithms. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
39. Lewine D. (1991). POSIX programmers guide. O'Reilly Media, Inc.
40. MacLean W.J., Sabihuddin S., Islam J. (2010) Leveraging cost matrix structure for hardware implementation of stereo disparity computation using dynamic programming. Computer Vision Image Understanding 114(11):1126–1138, DOI 10.1016/j.cviu.2010.03.011
41. Martnez del Rincon J., Thevenon J., Dieny R., Nebel J.C. Dense Pixel Matching Between Unrectified And Distorted Images Using Dynamic Programming. In International Conference on Computer Vision Theory and Applications, 24-26 February, Rome, Italy, 2012.
42. Mouser Electronics. 579-32MX795F512L80VF 32-bit Microcontroller. http://www.mouser.com.
43. Microchip (2011) MPLAB® X IDE. http://microchip.com/.
44. Nalpantidis L., Sirakoulis G.C., Gasteratos A. (2008) Review of stereo vision algorithms: from software to hardware. International Journal of Optomechatronics 2(4):435–462.
45. Needleman SB, Wunsch CD (1970) A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48:443–453.
46. Notredame C., Higgins D., Heringa J. (2000) T-Coffee: A novel method for multiple sequence alignments. Journal of Molecular Biology 302:205–217.
47. Ohta Y., Kanade T. (1985) Stereo by Intra- and Inter-Scanline Search using Dynamic Programming. Pattern Analysis and Machine Intelligence, IEEE Transactions on PAMI-7(2):139–154, DOI 10.1109/TPAMI.1985.4767639
48. Panda P.R., Catthoor F., Dutt N.D., Danckaert K., Brockmeyer E., Kulkarni C., Vandercappelle A., Kjeldsberg P.G. (2001) Data and memory optimization techniques for embedded systems. ACM Trans. Des. Autom. Electron. Syst. 6(2):149–206, DOI 10.1145/375977.375978
49. Pandaboard.org (2010) PandaBoard. http://pandaboard.org/.
50. Pandey J.G., Purushottam S., Karmakar A. Shekhar C. (2012) Platform-Based Extensible Hardware-Software Video Streaming Module for a Smart Camera System. In International Journal of Modeling and Optimization vol. 2, no. 4, pp. 482-487, 2012.
51. Raspberry Pi Foundation (2011) Raspberry Pi Board. http://www.raspberrypi.org/.
52. Raspberry Pi Foundation (2012) Raspbian OS. http://www.raspbian.org/.
53. Salmen J., Schlipsing M., Edelbrunner J., Hegemann S., Lüuke S. (2009) Real-Time Stereo Vision: Making more out of Dynamic Programming. In: Jiang

X., Petkov N. (eds) Computer Analysis of Images and Patterns, Lecture Notes in Computer Science, vol 5702, Springer Berlin Heidelberg, pp 1096–1103, DOI 10.1007/978-3-642-03767-2_133
54. Salvadori C., Petracca M., Bocchino S., Pelliccia R., Pagano P. (2014) A low-cost vehicle counter for next-generations ITS. In: Journal of Real Time Image Processing, DOI 10.1007/s11554-014-0411-4
55. Sarkar V. (2001) Optimized Unrolling of Nested Loops, vol 29, Springer, pp 545–581.
56. Scharstein D., Szeliski R. (2002) A taxonomy and evaluation of dense two-frame stereo correspondence algorithms, vol 47, pp 7–42.
57. Scuola Superiore Sant'Anna and Evidence s.r.l. (2011) SeedEye board. http://www.evidence.eu.com
58. Singh A.K., Kumar S.A. (2008) Microcontroller and Embedded System. New Age International (P) Limited.
59. Summit S. (1995) C programming FAQs: frequently asked questions. Addison Wesley Longman Publishing Inc., Redwood City, CA, USA.
60. Torr P.H.S., Criminisi A. (2004) Dense stereo using pivoted dynamic programming. Image and Vision Computing 22(10):795 – 806, DOI http://dx.doi.org/10.1016/j.imavis.2004.02.012, British Machine Vision Computing 2002.
61. Tseng Y.C., Chang N., Chang T.S. (2007) Low Memory Cost Block-Based Belief Propagation for Stereo Correspondence. In: Multimedia and Expo, 2007 IEEE International Conference on. pp 1415–1418, DOI 10.1109/ICME.2007.4284925
62. Veksler O. (2005) Stereo correspondence by dynamic programming on a tree. In: Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol 2, pp 384–390, DOI 10.1109/CVPR.2005.334
63. Vigliar M., Fratello M., Puglia L., Raiconi G. SASC: A hardware string alignment coprocessor for stereo correspondence. In Electronics Design, Systems and Applications (ICEDSA), 2012 IEEE International Conference, Kuala Lumpur, Malaysia, pp. 56-62, 5-6 Nov. 2012.
64. Wang L., Liao M., Gong M., Yang R., Nister D. (2006) High-Quality Real-Time Stereo using Adaptive Cost Aggregation and Dynamic Programming. In: 3D Data Processing, Visualization, and Transmission, Third International Symposium on. pp 798–805, DOI 10.1109/3DPVT.2006.75
65. Wei Y., Tsuhan C., Franchetti F., Hoe J. (2010) High Performance Stereo Vision Designed for Massively Data Parallel Platforms. Circuits and Systems for Video Technology, IEEE Transactions on 20(11):1509–1519, DOI 10.1109/TCSVT.2010.2077771
66. Zinner C., Humenberger M., Ambrosch K., Kubinger W. (2008) An Optimized Software-Based Implementation of a Census-Based Stereo Matching Algorithm. In: Bebis G., Boyle R., Parvin B., Koracin D., Remagnino P., Porikli F., Peters J., Klosowski J., Arns L., Chun Y., Rhyne T., Monroe L. (eds) Advances in Visual Computing, Lecture Notes in Computer Science, vol 5358, Springer Berlin Heidelberg, pp 216–227, DOI 10.1007/978-3-540-89639-5_21